JPA, REST and DataTransfer Objects

This exercise is very similar to what you did day-4, but with a few additional challenges added.

1) Start by cloning our suggested start code (from moodle) for projects that requires JPA and JAX-RS

This is the start code we assume you will use for the rest of this period and period2.

2) Create a local database for this exercise and change **persistence.xml** to point to this database.

3) Refactor (don't just rename) the entity class RenameMe.java into **BankCustomer.java**

4) Add the following fields (keep the existing id) to the **BankCustomer** Entity Class:

```
String firstName;
String lastName;
String accountNumber;
double balance;
int customerRanking;
String internalInfo;
```

***Info**: The fields marked with red, are meant as internal info to be used only by the bank. When you create your facade in the next step, some of the methods must prevent the red information from"coming out". Think of the Entity Class as your DataBase table, this represents your internal design, and often (as in almost always) we don't want to provide "outsiders" with all information from the database-tables.*

5) create a plain Java Class **MakeTestData.java** in the entity package. Add a main method, and use the entity framework to create a number of BankCustomers, and persist them to the database.
5a) Use Workbench to verify that these BankCustomers actually was persisted to the database

Using a **DTO class**

Here we will introduce a concept called Data Transfer Object (DTO)classes. It's a concept we will come back to MANY TIMES. For now, take it as a class used as a mapper when we create our JSON, that decouples the information that has to be sent, from the actual database design (the Entity Classes)

6) Create a class **CustomerDTO** in a package **dto** with (only) the following properties

```
int customerID;
String fullName;
String accountNumber;
double balance;
```

7) Add a constructor to the class that takes a BankCustomer instance and initializes the values using this instance (set customerID to the id used in the Entity Class)

8) Change the existing facade to provide the following methods.

```
CustomerDTO getCustomerByID(int id)
List<CustomerDTO> getCustomerByName(String name)
BankCustomer addCustomer(BankCustomer cust)
List<BankCustomer> getAllBankCustomers()
```

Assume that the last two methods only will be used by the bank and eventually will require users with elevated security privileges so here we use the Entity Class instead of a DTO, to simplify matters.

9) Create a **JUnit test**, to test the four methods.
Hint: In this week (only), use your existing database for the test.

10) Create a few REST endpoints as requested below
Rename the class `RenameMeResource.java` into **`BankCustomerResource.java`** and provide it with the following endpoints.

URL	**JSON to return** (with sample data added)
`/api/bankcustomer/{id}` *Return limited info for the requested customer*	`{ "customerID" : 100, "fullName" : "Kurt Wonnegut", "accountNumber" : "123-4567", "balance" :1000.25 }`

URL	**JSON to return** (with sample data added)
`/api/bankcustomer/all` *Return all information for all Customers.* **Hint:** Use the EntityClass as a mapper for your JSON-generation for this step	`[{ "id" : 100, "firstName" : "Kurt", "lastName" : "Wonnegut", "accountNumber" : "123-4567", "balance" :1000.25, "customerRanking": 3, "internalInfo" "bla bla bla" }, ... ]`

Deploy your solution

Use the guidelines given here to set up a droplet:
https://docs.google.com/document/d/1aGEcD9Jv_uIUgOUnpYiCCxvxnPbkeoaO7Bf7b70IAfA/edit?usp=sharing

and this video for the additional steps: https://youtu.be/qi47o74VwM8